(b) A formula is valid iff its negation is

- not valid
- also valid
- satisfiable
- not satisfiable

(c) Formula F is equivalent to formula G iff

- F IFF G is valid
- F IFF NOT(G) is not valid
- F XOR G satisfiable
- F XOR G is not satisfiable

Problem 3.11.
Indicate whether each of the following propositional formulas is valid (V), satisfiable but not valid (S), or not satisfiable (N). For the satisfiable ones, indicate a satisfying truth assignment.

$$
\begin{aligned}
&M \text{ IMPLIES } Q\\
&M \text{ IMPLIES } (\overline{P} \text{ OR } \overline{Q})\\
&M \text{ IMPLIES } [M \text{ AND } (P \text{ IMPLIES } M)]\\
(P \text{ OR } Q) &\text{ IMPLIES } Q\\
(P \text{ OR } Q) &\text{ IMPLIES } (\overline{P} \text{ AND } \overline{Q})\\
(P \text{ OR } Q) &\text{ IMPLIES } [M \text{ AND } (P \text{ IMPLIES } M)]\\
(P \text{ XOR } Q) &\text{ IMPLIES } Q\\
(P \text{ XOR } Q) &\text{ IMPLIES } (\overline{P} \text{ OR } \overline{Q})\\
(P \text{ XOR } Q) &\text{ IMPLIES } [M \text{ AND } (P \text{ IMPLIES } M)]
\end{aligned}
$$

Problem 3.12.
Show truth tables that verify the equivalence of the following two propositional formulas

$$
\begin{aligned}
&(P \text{ XOR } Q),\\
&\text{NOT}(P \text{ IFF } Q).
\end{aligned}
$$

Class Problems

Problem 3.16. **(a)** Verify by truth table that

$$(P \text{ IMPLIES } Q) \text{ OR } (Q \text{ IMPLIES } P)$$

is valid.

(b) Let P and Q be propositional formulas. Describe a single formula R using only AND's, OR's, NOT's, and copies of P and Q, such that R is valid iff P and Q are equivalent.

(c) A propositional formula is *satisfiable* iff there is an assignment of truth values to its variables—an *environment*—that makes it true. Explain why

P is valid iff NOT(P) is *not* satisfiable.

(d) A set of propositional formulas $P_1, \ldots, P_k$ is *consistent* iff there is an environment in which they are all true. Write a formula S such that the set $P_1, \ldots, P_k$ is *not* consistent iff S is valid.

Problem 3.17.
This problem[4] examines whether the following specifications are *satisfiable*:

1. If the file system is not locked, then...
   (a) new messages will be queued.
   (b) new messages will be sent to the messages buffer.
   (c) the system is functioning normally, and conversely, if the system is functioning normally, then the file system is not locked.
2. If new messages are not queued, then they will be sent to the messages buffer.
3. New messages will not be sent to the message buffer.

(a) Begin by translating the five specifications into propositional formulas using the four propositional variables:

$L ::=$ file system locked,
$Q ::=$ new messages are queued,
$B ::=$ new messages are sent to the message buffer,
$N ::=$ system functioning normally.

[4] Revised from Rosen, 5th edition, Exercise 1.1.36

It's easy to see that any efficient way of solving the circuit-SAT problem would yield an efficient way to solve the usual SAT problem for propositional formulas (Section 3.5). Namely, for any formula F, just construct a circuit C_F using that computes the values of the formula. Then there are inputs for which C_F gives output true iff F is satisfiable. Constructing C_F from F is easy, using a binary gate in C_F for each propositional connective in F. So an efficient circuit-SAT procedure leads to an efficient SAT procedure.

Conversely, there is a simple recursive procedure that will construct, given C, a formula E_C that is equivalent to C in the sense that the truth value E_C and the output of C are the same for every truth assignment of the variables. The difficulty is that, in general, the "equivalent" formula E_C, will be *exponentially larger* than C. For the purposes of showing that satifiability of circuits and satisfiability of formulas take roughly the same effort to solve, spending an exponential time translating one problem to the other swamps any benefit in switching from one problem to the other.

So instead of a formula E_C that is equivalent to C, we aim instead for a formula F_C that is "equisatisfiable" with C. That is, there will be input values that make C output **True iff** there is a truth assignment that satisfies F_C. (In fact, F_C and C need not even use the same variables.) But now we make sure that the amount of computation needed to construct F_C is not much larger than the size of the circuit C. In particular, the size of F_C will also not be much larger than C.

The idea behind the construction of F_C is that, given any digital circuit C with binary gates and one output, we can assign a distinct variable to each wire of C. Then for each gate of C, we can set up a propositional formula that represents the constraints that the gate places on the values of its input and output wires. For example, for an AND gate with input wire variables P and Q and output wire variable R, the constraint proposition would be

$$(P \text{ AND } Q) \text{ IFF } R. \tag{3.32}$$

(a) Given a circuit C, explain how to easily find a formula F_C of size proportional to the number of wires in C such that F_C is satisfiable iff C gives output **T** for some set of input values.

(b) Conclude that any efficient way of solving SAT would yield an efficient way to solve circuit-SAT.

Problem 3.25.

A 3-conjunctive normal form (3CNF) formula is a conjunctive normal form (CNF)

formula in which each OR-term is an OR of at most 3 *literals* (variables or negations of variables). Although it may be hard to tell if a propositional formula F is satisfiable, it is always easy to construct a formula $\mathcal{C}(F)$ that is

- a 3CNF,
- has at most 24 times as many occurrences of variables as F, and
- is satisfiable iff F is satisfiable.

Note that we do *not* expect $\mathcal{C}(F)$ to be *equivalent* to F. We do know how to convert any F into an equivalent CNF formula, and this equivalent CNF formula will certainly be satisfiable iff F is. But in many cases, the smallest CNF formula equivalent to F may be *exponentially larger* than F instead of only 24 times larger. Even worse, there may not be any **3**CNF equivalent to F.

To construct $\mathcal{C}(F)$, the idea is to introduce a different new variable for each operator that occurs in F. For example, if F was

$$((P \text{ XOR } Q) \text{ XOR } R) \text{ OR } (\overline{P} \text{ AND } S) \tag{3.33}$$

we might use new variables X_1, X_2, O and A corresponding to the operator occurrences as follows:

$$((P \underbrace{\text{XOR}}_{X_1} Q) \underbrace{\text{XOR}}_{X_2} R) \underbrace{\text{OR}}_{O} (\overline{P} \underbrace{\text{AND}}_{A} S).$$

Next we write a formula that constrains each new variable to have the same truth value as the subformula determined by its corresponding operator. For the example above, these constraining formulas would be

$$\begin{aligned} X_1 &\text{ IFF } (P \text{ XOR } Q),\\ X_2 &\text{ IFF } (X_1 \text{ XOR } R),\\ A &\text{ IFF } (\overline{P} \text{ AND } S),\\ O &\text{ IFF } (X_2 \text{ OR } A) \end{aligned}$$

(a) Explain why the AND of the four constraining formulas above along with a fifth formula consisting of just the variable O will be satisfiable iff (3.33) is satisfiable.

(b) Explain why each constraining formula will be equivalent to a 3CNF formula with at most 24 occurrences of variables.

(c) Using the ideas illustrated in the previous parts, briefly explain how to construct $\mathcal{C}(F)$ for an arbitrary propositional formula F. (No need to fill in all the details for this part—a high-level description is fine.)

Problem 3.31.
Find a counter-model showing the following is not valid.

$$[\exists x.\, P(x) \text{ AND } \exists x.Q(x)] \text{ IMPLIES } \exists x.[P(x) \text{ AND } Q(x)]$$

(Just define your counter-model. You do not need to verify that it is correct.)

Problem 3.32.
Which of the following are *valid*? For those that are not valid, desribe a counter-model.

(a) $\exists x \exists y.\ P(x, y)$ IMPLIES $\exists y \exists x.\ P(x, y)$

(b) $\forall x \exists y.\ Q(x, y)$ IMPLIES $\exists y \forall x.\ Q(x, y)$

(c) $\exists x \forall y.\ R(x, y)$ IMPLIES $\forall y \exists x.\ R(x, y)$

(d) NOT$(\exists x\ S(x))$ IFF $\forall x$ NOT$(S(x))$

Problem 3.33. **(a)** Verify that the propositional formula

$$(P \text{ IMPLIES } Q) \text{ OR } (Q \text{ IMPLIES } P)$$

is valid.

(b) The valid formula of part (a) leads to sound proof method: to prove that an implication is true, just prove that its converse is false.[5] For example, from elementary calculus we know that the assertion

> If a function is continuous, then it is differentiable

is false. This allows us to reach at the correct conclusion that its converse,

> If a function is differentiable, then it is continuous

is true, as indeed it is.
But wait a minute! The implication

> If a function is differentiable, then it is not continuous

is completely false. So we could conclude that its converse

[5] This problem was stimulated by the discussion of the fallacy in [4].

(e) How could you express "Everyone except for Claire likes Emily" using just propositional connectives *without* using any quantifiers $(\forall, \exists)$? Can you generalize to explain how *any* logical formula over this domain of discourse can be expressed without quantifiers? How big would the formula in the previous part be if it was expressed this way?

Problem 3.35.
For each of the logical formulas, indicate whether or not it is true when the domain of discourse is $\mathbb{N}$, (the nonnegative integers 0, 1, 2, ...), $\mathbb{Z}$ (the integers), $\mathbb{Q}$ (the rationals), $\mathbb{R}$ (the real numbers), and $\mathbb{C}$ (the complex numbers). Add a brief explanation to the few cases that merit one.

$$\begin{aligned}
\exists x.\, x^2 &= 2 \\
\forall x. \exists y.\, x^2 &= y \\
\forall y. \exists x.\, x^2 &= y \\
\forall x \neq 0. \exists y.\, xy &= 1 \\
\exists x. \exists y.\, x + 2y = 2 \ &\text{AND}\ 2x + 4y = 5
\end{aligned}$$

Problem 3.36.
The goal of this problem is to translate some assertions about binary strings into logic notation. The domain of discourse is the set of all finite-length binary strings: λ, 0, 1, 00, 01, 10, 11, 000, 001, ... (Here λ denotes the empty string.) In your translations, you may use all the ordinary logic symbols (including =), variables, and the binary symbols `0`, `1` denoting 0, 1.

A string like $\texttt{01}x\texttt{0}y$ of binary symbols and variables denotes the *concatenation* of the symbols and the binary strings represented by the variables. For example, if the value of x is `011` and the value of y is `1111`, then the value of $\texttt{01}x\texttt{0}y$ is the binary string `0101101111`.

Here are some examples of formulas and their English translations. Names for these predicates are listed in the third column so that you can reuse them in your solutions (as we do in the definition of the predicate NO-1S below).

Meaning	Formula	Name
x is a prefix of y	$\exists z\ (xz = y)$	PREFIX(x, y)
x is a substring of y	$\exists u \exists v\ (uxv = y)$	SUBSTRING(x, y)
x is empty or a string of 0's	NOT(SUBSTRING$(1, x)$)	NO-1S(x)

(a) x consists of three copies of some string.

(b) x is an even-length string of 0's.

(c) x does not contain both a 0 and a 1.

(d) x is the binary representation of $2^k + 1$ for some integer $k \geq 0$.

(e) An elegant, slightly trickier way to define NO-1S(x) is:

$$\text{PREFIX}(x, 0x). \tag{*}$$

Explain why (*) is true only when x is a string of 0's.

Problem 3.37.
In this problem we'll examine predicate logic formulas where the domain of discourse is $\mathbb{N}$. In addition to the logical symbols, the formulas may contain ternary predicate symbols A and M, where

$$A(k, m, n) \text{ means } k = m + n,$$
$$M(k, m, n) \text{ means } k = m \cdot n.$$

For example, a formula "Zero(n)" meaning that n is zero could be defined as

$$\text{Zero}(n) ::= A(n, n, n).$$

Having defined "Zero," it is now OK to use it in subsequent formulas. So a formula "Greater(m, n)" meaning $m > n$ could be defined as

$$\text{Greater}(m, n) ::= \exists k.\ \text{NOT}(\text{Zero}(k)) \text{ AND } A(m, n, k).$$

This makes it OK to use "Greater" in subsequent formulas.

Write predicate logic formulas using only the allowed predicates A, M that define the following predicates:

(a) Equal(m, n) meaning that $m = n$.

(b) One(n) meaning that $n = 1$.

(c) $n = i(m \cdot j + k^2)$

(d) Prime(p) meaning p is a prime number.

(e) Two(n) meaning that $n = 2$.

The results of part (e) will entend to formulas $\text{Three}(n), \text{Four}(n), \text{Five}(n), \ldots$ which are allowed from now on.

(f) $\text{Even}(n)$ meaning n is even.

(g) (Goldbach Conjecture) Every even integer $n \geq 4$ can be expressed as the sum of two primes.

(h) (Fermat's Last Theorem) Now suppose we also have

$$X(k, m, n) \quad \text{means } k = m^n.$$

Express the assertion that there are no positive integer solutions to the equation:

$$x^n + y^n = z^n$$

when $n > 2$.

(i) (Twin Prime Conjecture) There are infinitely many primes that differ by two.

Homework Problems

Problem 3.38.
Express each of the following predicates and propositions in formal logic notation. The domain of discourse is the nonnegative integers, $\mathbb{N}$. Moreover, in addition to the propositional operators, variables and quantifiers, you may define predicates using addition, multiplication, and equality symbols, and nonnegative integer *constants* (`0`, `1`,...), but no *exponentiation* (like x^y). For example, the predicate "n is an even number" could be defined by either of the following formulas:

$$\exists m.\ (2m = n), \qquad \exists m.\ (m + m = n).$$

(a) m is a divisor of n.

(b) n is a prime number.

(c) n is a power of a prime.

Problem 3.39.
Translate the following sentence into a predicate formula:

> There is a student who has e-mailed at most two other people in the class, besides possibly himself.